Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Elettronica

Realizzazione di Encoder BCH DVB-T2 su FPGA

Relatore: Prof. Daniele Vogrig

Laureando: Fabio De Rossi

Anno accademico 2009/2010

Riassunto

Questa tesina documenta la realizzazione di un Encoder BCH compatibile con lo standard DVB-T2 su una FPGA della famiglia Xilinx Virtex 4.
Lo sviluppo tecnologico ha portato al passaggio dalla televisione analogica a quella digitale, rendendo così necessari metodi di rivelazione e correzione degli errori che garantiscano alte prestazioni. Il modulatore di un sistema di trasmissione DVB-T2 utilizza, in successione, le codifiche BCH e LDPC. I codici BCH appartengono alla classe dei codici lineari ciclici sistematici con capacità di correzione multipla, e nella presente realizzazione possono correggere fino a 10 o 12 errori, dipendentemente dalla modalità di codifica adottata.
Essendo ciclici i codici BCH permettono l'implementazione del proprio codificatore mediante l'impiego di un LFSR (*Linear Feedback Shift Register*) di Galois opportunamente configurato. È questa infatti la struttura principale attorno alla quale si è sviluppato l'intero encoder, che comprende anche la gestione di due linee di segnalazione della disponibilità dei dati di ingresso e uscita, una linea di reset, una di abilitazione del componente, una linea che segnala lo stato dell'encoder ed infine un ingresso che consente di selezionare la modalità di codifica fra quelle previste dallo standard DVB-T2.
Per poter verificare il corretto funzionamento dell'encoder, oltre ad un programma di *testbench*, sono state create alcune applicazioni con il linguaggio *Matlab*. Nell'ultimo capitolo sono riportate delle schermate tratte dal tool di simulazione *ISim* che evidenziano i punti di funzionamento di maggior interesse e vengono discussi i risultati di tali test.

Indice

Capitolo 1

Introduzione

1.1 Televisione analogica

I tradizionali servizi televisivi fanno uso della tecnologia analogica per poter provvedere alla diffusione di segnali audio e video. Tali segnali derivano da sorgenti come videocamere e microfoni o da sistemi di memorizzazione e elaborazione come registratori e computer, vengono passati a differenti modulatori, le cui uscite vengono poi multiplexate ed innalzate in frequenza per essere trasmesse.
Per motivi storici, per la televisione analogica sono previsti vari metodi di modulazione, multiplexing e diverse frequenze alle quali inviare il segnale a specifici mezzi trasmissivi. I tre maggiori standard sono [5]:

- NTSC (*National Television System Committee*) Usato principalmente negli stati uniti d'America e nell'America del Sud.
- SECAM (*Systeme Electronique (pour) Couleur avec Memoire*) Usato in Francia e in paesi dell'Europa dell'est come Polonia e Russia.
- PAL (*Phase Alternating Line*) Diffuso in molti altri paesi, inclusa l'Europa occidentale e l'Australia.

Il segnale video che appare su un televisore è creato da una sequenza di 25 o 30 immagini, o *frame*, al secondo (si parla di frequenza di aggiornamento di 25Hz e 35Hz rispettivamente). Ogni immagine consiste di un certo numero di linee che vengono stampate dalla sinistra alla destra e dall'alto al basso dello schermo, più altre linee che contengono dati informativi, il cui primo scopo era di dare al tubo a raggio catodico dei televisori CRT il tempo di tornare dal fondo destro del display alla fine di un frame alla sommità sinistra all'inizio del frame successivo (*Vertical Blanking Interval*).
Il segnale audio delle trasmissioni televisive analogiche ha una banda di circa 15 kHz. Originalmente veniva trasmesso un solo canale monofonico, che è stato poi esteso a due canali indipendenti, detti stereo.

1.2 DTV

La televisione digitale è la naturale evoluzione della televisione analogica. Fino a qualche anno fa infatti il segnale televisivo veniva rappresentato in formato analogico in tutte le sue fasi, dalla sorgente (telecamere e microfoni) attraverso la fase di editing, di trasmissione (condizionamento, modulazione, amplificazione), fino alla ricezione (demodulazione del segnale da parte dell'apparecchio televisivo). Con lo sviluppo delle tecnologie si è però passati all'uso di strumenti di registrazione digitali e all'editing per mezzo di software, per giungere infine alla completa digitalizzazione dell'intero processo di telecomunicazione dell'informazione televisiva; si parla pertanto di DTV (*Digital TeleVision*).

Il segnale digitale vanta una maggiore qualità audio e video rispetto al predecessore analogico, ed offre inoltre un angolo di visuale più ampio (*panoramic screen*) con un maggiore grado di risoluzione. La scansione dell'immagine può essere interlacciata, ossia viene alternata la stampa delle righe pari e dispari, oppure progressiva, nel qual caso le righe vengono stampate in ordine. La qualità dell'immagine viene indicata con sigle come "720p", dove 720 è il numero di righe e la "p" indica l'uso della scansione progressiva, mentre in altri casi la "i" sta per interlacciata.

Gli standard video utilizzati dalla televisione digitale sono [3]:

- *Standard-Definition Television* (SDTV) prevede una risoluzione di 480 o 576 righe. Il formato può essere 16:9 o 4:3.
- *High-Definition Television* (HDTV) tale formato offre una qualità maggiore rispetto allo SDTV. I formati HDTV più popolari sono lo 720p e il 1080i.
- *Enhanced-Definition Television* (EDTV) la cui qualità si situa tra la SDTV e l'HDTV.
- *Low-Definition Television* (LDTV) la cui qualità è inferiore a quella dello standard SDTV. Si riferisce a trasmissioni con risoluzione uguale o simile ai sistemi TV analogici a bassa definizione.

1.3 Standard della DTV

I moderni sistemi di televisione digitale fanno tutti parte di uno dei due seguenti set di standard [5]:

- *American Advanced Television Systems Committee* (ATSC) Sistema americano.
- *Digital Video Broadcast* (DVB) Usato nella maggior parte del resto del mondo, che include Europa, buona parte dell'Asia ed Australia.

La trasmissione del segnale televisivo digitale può avvenire per via aerea (terrestre), satellitare o via cavo. Relativamente a tali modalità la DVB definisce i seguenti principali standard:

- **DVB-T** (Digital Video Broadcasting - Terrestrial) la ricezione del segnale avviene per mezzo delle tradizionali antenne televisive.
- **DVB-S** (Digital Video Broadcasting - Satellite) il segnale è trasmesso via satellite e viene ricevuto da opportune antenne paraboliche.
- **DVB-C** (Digital Video Broadcasting - Cable) il segnale viene trasmesso tramite un cavo coassiale.
- **DVB-H** (Digital Video Broadcasting - Handled) è uno standard derivato dal DVB-T creato per la trasmissione di segnali televisivi, radio e contenuti multimediali a dispositivi portatili come palmari e smartphone.

1.4 Standard DVB-T2

Il DVB-T2 (Second Generation Digital Terrestrial Video Broadcasting) è nato come estensione del precedente standard per la televisione digitale terrestre DVB-T. I principali scopi del nuovo standard, in riferimento al precedente, sono l'incremento da un minimo del 30% a circa il 50% del data rate ed un incremento della robustezza per consentire la ricezione da parte di applicazioni mobili. Il DVB-T2 comunque è stato realizzato cercando di non reinventare, per quanto possibili, le soluzioni tecnologiche previste dai precedenti standard DVB. Per questo, il T2 ha adottato due fondamentali tecnologie del DVB-S2 [2]:

- L'architettura a livelli (Layers), e in particolare l'inpacchettamento dei dati in Frame in banda base (Baseband Frame - `BBFRAME`).
- Lo stesso sistema di correzione degli errori (FEC - Forward Error Correction), composto essenzialmente da un codificatore BCH seguito da un codificatore LDPC e da un Bit Interleaver.

Comunque, per via della differente natura della modulazione per la trasmissione terrestre che è basata sulla tecnica COFDM con intervallo di guardia, nel T2 sono state incluse nuove tecniche di bit-interleaving e constellation-mapping. Per poter massimizzare la capacità di trasmissione è stato introdotto un gran numero di opzioni configurabili, ad esempio la grandezza delle FFT, la frazione dell'intervallo di guardia, i pilot patterns.

1.4.1 Modello architetturale

Il diagramma a blocchi di un sistema di ricetrasmissione DVB-T2 è visibile in figura 1.

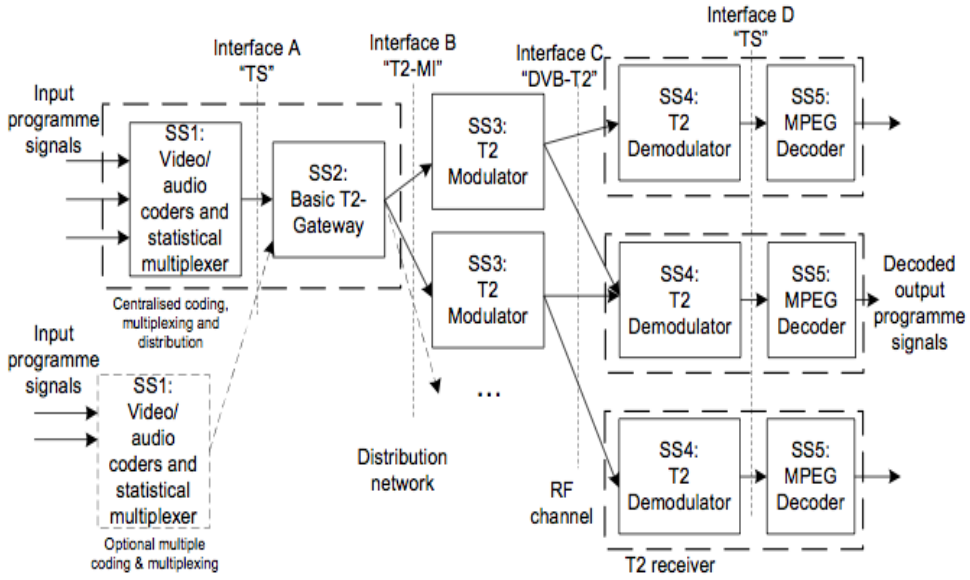

Fig. 1.1: Diagramma a blocchi di un sistema DVB-T2 [2].

L'intero sottosistema può essere diviso nei seguenti cinque sottosistemi fondamentali, in primi tre dei quali fanno parte del lato trasmissione, gli ultimi due del lato ricezione:

- SS1: Sottosistema di codifica e multiplexing. I segnali in entrata a questo blocco vengono codificati secondo lo standard MPEG-2, ottenendo un PLP (Physical Layer Pipe). Il DVB-T2 è orientato alla trasmissione, oltre che ai flussi di dati audio/video (TS, Transport Streams), a flussi di dati generici. I segnali codificati vengono quindi opportunamente multiplexati in un flusso unico di dati.

- SS2: Basic T2-Gateway. La funzionalità del Basic T2-Gateway include tutte quelle operazioni relative al physical-layer (livello fisico) dello standard che non sono strettamente obbligatorie. In uscita al sottosistema si ha una sequenza di pachetti che può essere passata a uno o più modulatori in un network.

- SS3: Sottosistema di modulazione. Usa i `BBFRAME` e le istruzioni contenute nei frame ricevuti in ingresso per produrre in uscita frame T2 e trasmetterli con un timing appropriato per la corretta sincronizzazione.

- SS4: Sottosistema di demodulazione. Riceve un segnale a radio frequenza da uno o più trasmettitori, esegue la demodulazione dei pacchetti e fornisce un singolo flusso di dati in uscita.

- SS5: Sottosistema di decodifica. Preleva il flusso demodulato nel precedente sottosistema ed eseguendone la decodifica estrae i segnali informativi audio e video.

1.4.2 Struttura dei frame

I `BBFRAME` sono l'unità base nella struttura logica del DVB-T2. Oltre ai bit informativi possono contenere un header e dei bit di *padding*, nel caso i dati fossero insufficienti. La lunghezza totale di ogni `BBFRAME` è costante in riferimento ad una dato code rate per la codifica LDPC.
I frame in bandabase vengono trattati come dei semplici messaggi ai quali sono applicate le codifiche BCH e LDPC. La parola di codice risultante è detta `FECFRAME` e può essere composta da 64800 (Long `FECFRAME`) o da 16200 (Short `FECFRAME`) bit.

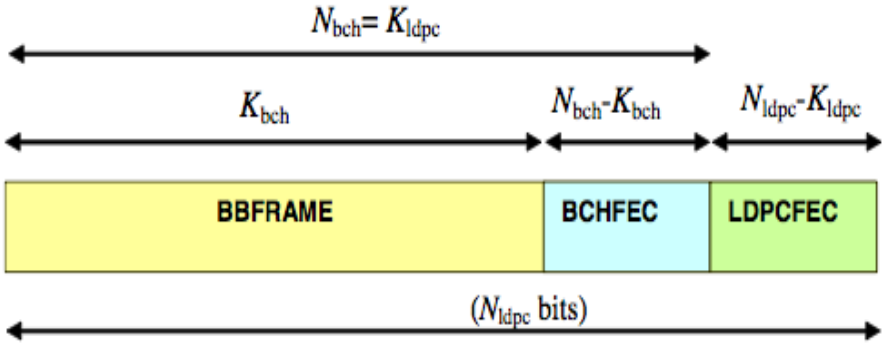

Fig. 1.2: Struttura di un `FECFRAME` [1].

1.4.3 FEC

Il sistema di protezione dagli errori è lo stesso previsto dallo standard DVB-S2, ovvero un encoder BCH e un encoder LDPC seguiti da un Bit Interleaver. All'interno del modulatore DVB-T2 i Baseband Frame comprendenti il Baseband Header e il padding-block sono inviati al blocco di FEC, dove vengono appesi in coda al frame prima i bit di correzione BCH e successivamente quelli LDPC che hanno un code rate selezionabile. Il pacchetto risultante viene detto `FECFRAME`.

Codifica BCH
La codifica BCH ha il compito di proteggere la trasmissione dagli errori ad alti rapporti C/N (carrier-to-noise ratio). La capacità di correzione degli errori è di 10 o 12 bit a seconda del code rate dell'LDPC. Tale codifica verrà esaminata approfonditamente in un successivo capitolo, in quanto costituisce esattamente il sistema che il presente lavoro ha implementato.

Codifica LDPC
La LDPC è una codifica molto potente e al contempo molto complessa, che può essere implementata con architetture munite di un certo grado di parallelismo. Nello standard DVB-T2 i 6 code rate previsti, 1/2, 3/5, 2/3, 3/4, 4/5 e 5/6 sono definiti sia per i `FECFRAME` normali che per quelli brevi. Il code rate è il rapporto tra il numero di bit che vengono codificati (K_{ldpc}, in riferimento alla figura 1.2) e la lunghezza del pacchetto di uscita (N_{ldpc}).

Bit Interleaver
I codici LDPC dello standard DVB-T2 sono irregolari e pertanto il livello di protezione di ogni bit non è uniforme. Nemmeno la protezione tra i bit in un simbolo di una constellazione multilivello è uniforme ma varia nella corrispondenza tra i bit di codice e i bit della costellazione. Il compito del bit interleaver è quello di ridisporre i dati contenuti nei `FECFRAME` per massimizzare tale corrispondenza ed ottimizzare la modulazione.

1.4.4 Modulazione

Come per lo standard DVB-T il metodo di modulazione usato è di tipo COFDM (Coded Orthogonal Frequency Division Multiplexing) con intervallo di guardia, che trasporta i dati su molte portanti ortogonali. Il numero di tali portanti è caratterizzato dalla dimensione della FFT nel modulatore e i valori possibili sono 1K, 2K, 4K, 8K, 16K e 32K. Essendo la larghezza di banda complessiva fissa, all'aumentare delle portanti si ha una riduzione della spaziatura tra di esse ed un incremento del periodo di simbolo. Per quanto piccole dimensioni della FFT garantiscano maggiore robustezza per trasmissioni in condizioni critiche, con FFT di elevate dimensioni si ottiene una maggiore robustezza nei confronti del rumore impulsivo e inferiori livelli di potenza fuori banda. I tipi di modulazione possibili sono:

- QPSK
- 16-QAM
- 64-QAM
- 256-QAM

Mentre le prime tre erano previste già dallo standard DVB-T, l'ultima è un'innovazione del DVB-T2. L'uso della 256-QAM permette di avere 8

bit per simbolo, che rispetto ai 6 bit per simbolo della 64-QAM fornisce un incremento del 33% dell'efficienza spettrale. Un'ulteriore innovazione consiste nella possibilità di ruotare la costellazione della modulazione in uso a sinistra di un certo angolo. Le modulazioni QAM, infatti, equivale alla modulazione di ampiezza di due sinusoidi in quadratura, ovvero sfasate di 90 gradi, il che equivale a modulare sia in ampiezza che in fase una unica ideale sinusoide. Il segnale modulato è esprimibile come

$$s(t) = I(t)cos(2\pi f_o t) - Q(t)sin(2\pi f_o t)$$

dove

$$I(t) = Acos(\phi) \text{ e } Q(t) = Asin(\phi)$$

sono i segnali modulanti. Si usa perciò rappresentare le costellazioni QAM sul piano complesso I-Q. Grazie alla rotazione della costellazione (detta perciò anche Q-delayed) le due componenti I e Q vengono separate dal bit interleaver cosicché possano viaggire a frequenze e tempi differenti. Il vantaggio è che, se il canale di trasmissione distrugge una delle componenti, l'altra può essere usata per recuperare l'informazione.

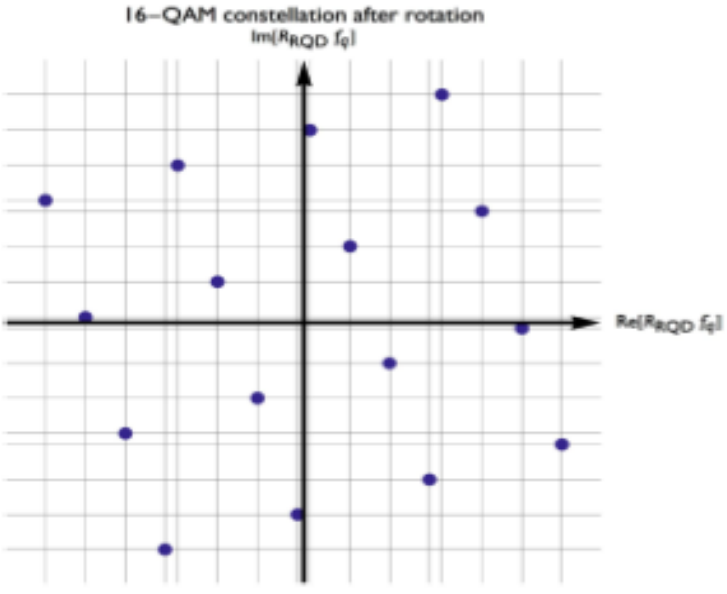

Fig. 1.3: Costellazione 16-QAM ruotata [2].

Capitolo 2

Codici correttori

2.1 Introduzione

Lo scopo dei canali di comunicazione consiste nella trasmissione di segnali informativi, facendo sì che in uscita al blocco di ricezione si ottenga esattamente lo stesso segnale presente alla sorgente del sistema. In un qualsiasi canale reale, infatti, il segnale passante è soggetto alla presenza di rumore che ne altera l'andamento, impedendone la corretta ricezione. Per questo hanno grande importanza i metodi di codifica dei segnali che modificano, mediante un particolare algoritmo, il segnale all'ingresso del canale di comunicazione allo scopo di renderlo meno corruttibile da rumore e interferenze. Simmetricamente, al ricevitore viene operata la decodifica del segnale ricevuto per ricavare l'informazione originale.
Le strategie che guidano la realizzazione delle codifiche sono due, per questo si distingue tra [4]:

- **codifica di sorgente**, il cui obiettivo primario è la compressione del segnale informativo mediante l'eliminazione della ridondanza.
- **codifica di canale**, che consente la rivelazione e/o correzione degli errori per mezzo dell'aggiunta di dati di ridondanza, che non hanno valore informativo.

2.2 Codifica di canale

Ci sono due modi di usare i bit di ridondanza introdotti da una codifica di canale [6]:

Automatic Repeat reQuest (ARQ)
: Permettono di *rivelare* ma non correggere eventuali errori attraverso l'impiego della ridondanza. Quando si hanno uno o più errori il ricevitore

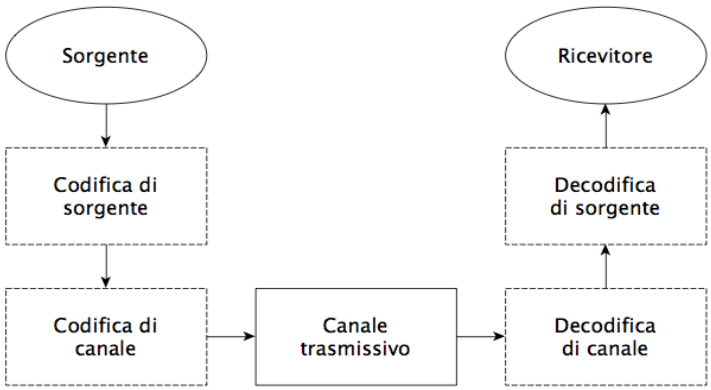

Fig. 2.1: Schema a blocchi di un sistema trasmissivo.

invia una richiesta di ritrasmissione dei dati al trasmettitore attraverso un canale dedicato (*feedback channel*). Tale operazione può essere ripetuta fino alla corretta ricezione dei dati, in qual caso si riprende normalmente la trasmissione.

Forward Error Correction (FEC)
Non vi è la possibilità di chiedere la ritrasmissione dei dati, pertanto al ricevitore viene implementato un algoritmo di correzione che sfrutta i dati di ridondanza per *rivelare e correggere* eventuali errori presenti nel messaggio. Tale operazione è però possibile solo se il numero di errori non supera la capacità di correzione del codice, pena il rischio di introdurne di ulteriori.

Un'altra fondamentale distinzione dei codici di canale viene operata fra i *codici a blocco*, per i quali la codifica è una funzione istantanea della parola informativa, e i *codici convoluzionali* la cui codifica dipende invece da un determinato numero di cifre di informazione precedenti.

2.3 Codici a blocco

I codici a blocco operano dividendo il segnale da trasmettere, presente in ingresso in forma binaria, in blocchi di k bit, detti bit di *messaggio* [7]. L'alfabeto d'ingresso è quindi costituito da 2^k differenti messaggi. L'encoder fornisce in uscita blocchi di n bit, detti *parole di codice* (*codeword*), ovvero aggiunge n - k bit detti di ridondanza o di controllo di parità. Naturalmente spetta al decodificatore il compito di utilizzare tali bit al fine di rilevare e correggere eventuali errori. Un codice a blocco così definito viene denotato come $C_b(n, k)$

Il rapporto (code rate) $R_c = k/n$ è una misura del livello di ridondanza applicata, ed è inoltre strettamente legato all'aumento di banda di frequenza necessario per la trasmissione in confronto al caso di segnale non codificato. Dal momento che i 2^k possibili messaggi sono convertiti in parole di codice di n bit, la procedura di codifica può essere assimilata ad una espansione dello spazio vettoriale dei messaggi di dimensione 2^k in uno spazio di dimensione 2^n, del quale solo 2^k vettori, convenientemente scelti, possono essere selezionati.
I codici a blocco possono quindi essere opportunamente analizzati utilizzando la teoria degli spazi vettoriali. Si riassume perciò quanto appena detto:

Un blocco è rappresentato da una k-upla $\mathbf{m} = (m_1, m_2, \ldots, m_k)$ chiamata *messagio*. L'encoder trasforma ogni possibile messaggio (ve ne sono 2^k per un codice binario) in una n-upla $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ chiamata *parola di codice*, aggiungendo ridondanza di lunghezza $n - k$. L'insieme delle 2^k parole di codice di lunghezza n viene chiamato *codice di blocco* di tipo (n, k).

Vi sono varie categorie di codici a blocchi, ad esempio i codici ciclici e i codici lineari, dei quali i codici BCH fanno parte.

2.4 Codici lineari

Premessa: L'alfabeto di un codice lineare è un campo finito q-ario, detto anche campo di Galois, denominato $\mathbb{F}_q$, ovvero un insieme finito e non vuoto di elementi sul quale sono definite le operazioni di addizione $(+)$ e moltiplicazione $(*)$ fra elementi interni del campo e il cui risultato è sempre un elemento appartenente al campo stesso.
La maggioranza dei codici lineari opera sul campo binario $\mathbb{F}_2$, i cui elementi sono $[0, 1]$, e le relative potenze $\mathbb{F}_2^n$. I codici BCH sono fra questi.

Definizione 2.1 (Codice lineare)**.** Un codice a blocco di lunghezza n provvisto di q^k parole di codice viene definito *lineare* di tipo (n, k) su $\mathbb{F}_q$ se e solo se le q^k parole formano uno sottospazio k-dimensionale di $\mathbb{F}_q^n$.

Come conseguenza della sua definizione, un codice a blocco lineare è caratterizzato dal fatto che la somma di due parole di codice qualsiasi è ancora una parola di codice.

2.4.1 Matrice generatrice

Essendo un codice a blocco lineare $C_b(n, k)$ un sottospazio dello spazio vettoriale $\mathbb{F}_q^n$, ci saranno k vettori linearmente indipendenti che fanno parte

delle parole di codice detti $\mathbf{g_0}, \mathbf{g_1}, \ldots, \mathbf{g_{k-1}}$, tali che ogni possibile parola di codice sia una loro combinazione lineare [6]:

$$\mathbf{c} = m_0 \cdot \mathbf{g_0} + m_1 \cdot \mathbf{g_1} + \ldots + m_{k-1} \cdot \mathbf{g_{k-1}} \tag{2.1}$$

Tali vettori linearmente indipendenti formano una matrice detta matrice generatrice **G**:

$$\mathbf{G} = \begin{bmatrix} \mathbf{g_0} \\ \mathbf{g_1} \\ \vdots \\ \mathbf{g_{k-1}} \end{bmatrix} = \begin{bmatrix} g_{00} & g_{01} & \cdots & g_{0,n-1} \\ g_{10} & g_{11} & \cdots & g_{1,n-1} \\ \vdots & \vdots & & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{bmatrix} \tag{2.2}$$

L'utilità della matrice generatrice sta nel fornire la parola di codice voluta a partire da un messaggio $\mathbf{m} = (m_0, m_1, \ldots, m_{k-1})$ attraverso la moltiplicazione matriciale

$$\begin{aligned} \mathbf{c} = \mathbf{mG} &= (m_0, m_1, \ldots, m_{k-1}) \begin{bmatrix} g_{00} & g_{01} & \cdots & g_{0,n-1} \\ g_{10} & g_{11} & \cdots & g_{1,n-1} \\ \vdots & \vdots & & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{bmatrix} = \\ &= m_0 \cdot \mathbf{g_0} + m_1 \cdot \mathbf{g_1} + \ldots + m_{k-1} \cdot \mathbf{g_{k-1}} \end{aligned} \tag{2.3}$$

Ciò rende evidente che la matrice generatrice definisce completamente un codice lineare.

2.4.2 Codici a blocco lineari sistematici

Un codice a blocco lineare è detto *sistematico* se ogni sua parola di codice consiste di due parti, la parte di *messaggio* e la parte di *controllo di parità*. La parte di messaggio riproduce senza alterazioni i k bit del messaggio mentre la parte di controllo di parità contiene gli $n - k$ bit di ridondanza, come visibile in figura 2.2.

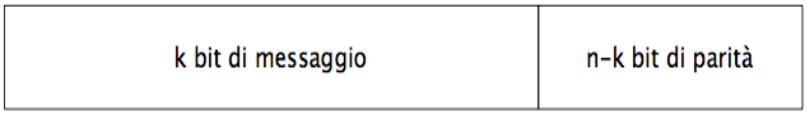

Fig. 2.2: Parola di codice di un codice sistematico.

Un codice sistematico è completamente specificato da una matrice generatrice $k \times n$, ottenibile riducendo in forma canonica la matrice generatrice $\mathbf{G}$, della forma

$$\mathbf{G} = \begin{bmatrix} \mathbf{g_0} \\ \mathbf{g_1} \\ \vdots \\ \mathbf{g_{k-1}} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 & p_{00} & p_{01} & \cdots & p_{0,n-k-1} \\ 0 & 1 & \cdots & 0 & p_{10} & p_{11} & \cdots & p_{1,n-k-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n-k-1} \end{bmatrix} \tag{2.4}$$

Indicando con $\mathbf{I_k}$ la matrice d'identità $k \times k$ e con $\mathbf{P}$ una matrice di dimensioni $k \times (n-k)$, si può quindi scrivere

$$\mathbf{G} = [\mathbf{I_k}\, \mathbf{P}] \tag{2.5}$$

2.4.3 Matrice di controllo di parità

Un codice lineare di tipo (n, k) può essere specificato anche da una matrice $\mathbf{H}$ di dimensioni $(n-k) \times n$. Si ha che una n-upla $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ è una parola di codice se e solo se

$$\mathbf{cH^T} = \mathbf{0} \tag{2.6}$$

ovvero se il prodotto interno di $\mathbf{c}$ e di ogni riga di $\mathbf{H}$ è nullo. Tale matrice $\mathbf{H}$ è detta *matrice di controllo di parità*.
Nel caso di un codice lineare sistematico di tipo (n, k) con matrice generatrice $\mathbf{G} = [\mathbf{I_k}\, \mathbf{P}]$, la matrice di controllo di parità è esprimibile come

$$\mathbf{H} = [\mathbf{I_{n-k}} \mathbf{P^T}] \tag{2.7}$$

dove $\mathbf{I_{n-k}}$ è la matrice di identità di dimensioni $(n-k) \times (n-k)$ e $\mathbf{P^T}$ è la trasposta di $\mathbf{P}$.

2.4.4 Cenni sulla decodifica

Denotando con $\mathbf{r} = r_0, r_1, \ldots, r_{n-1}$ il vettore ricevuto dal decodificatore, si ha che $\mathbf{r}$ è esprimibile come la somma nel campo binario $\mathbb{F}_2$ della parola di codice $\mathbf{c}$ e di un *vettore di errore* $\mathbf{e} = e_0, e_1, \ldots, e_{n-1}$ introdotto dal canale:

$$\mathbf{r} = \mathbf{c} + \mathbf{e} \tag{2.8}$$

Definizione 2.2 (Sindrome)**.** Si definisce *sindrome* di un vettore $\mathbf{r} \in \mathbb{F}_q^n$ rispetto alla matrice di controllo di parità $\mathbf{H}$ il vettore $\mathbf{s}$ definito come

$$\mathbf{s} = \mathbf{rH^T} = (s_0, s_1, \ldots, s_{n-k-1}) \tag{2.9}$$

Per verificare se un vettore $\mathbf{r}$ ricevuto contenga errori di trasmissione ne si calcola pertanto la sindrome. Se questa risulta essere un vettore nullo, infatti, dalla (2.6) è evidente che $\mathbf{r}$ corrisponde alla parola di codice trasmessa. Inoltre, dalla (2.8) si ricava

$$\mathbf{s} = \mathbf{rH^T} = (\mathbf{c} + \mathbf{e})\mathbf{H^T} = \mathbf{cH^T} + \mathbf{eH^T} = \mathbf{eH^T} \tag{2.10}$$

Quando la sindrome contiene almeno un elemento non nullo viene segnalata la presenza di errori nel vettore ricevuto. C'è comunque la possibilità che la sindrome sia nulla nonostante ci siano degli errori. Ciò avviene quando il vettore di errore sia anch'esso una parola di codice; in tal caso è impossibile rilevare la presenza e quindi correggere tali errori.

2.4.5 Distanza minima

La distanza minima d_{min} è un parametro importante per un codice correttore. Prima della sua definizione è utile introdurre i concetti di peso e distanza di Hamming [6].

Definizione 2.3 (Peso di Hamming)**.** Il numero di componenti non nulli $c_i \neq 0$ di un dato vettore $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ è chiamato peso di Hamming di tal vettore ed è denotato come $w(\mathbf{c})$. Nel caso di un vettore definito sul campo binario $\mathbb{F}_2$ il peso di Hamming è uguale al numero di '1' nel vettore.

Definizione 2.4 (Distanza di Hamming)**.** La distanza di Hamming tra due vettori $\mathbf{c_1} = (c_{01}, c_{11}, \ldots, c_{n-1,1})$ e $\mathbf{c_2} = (c_{02}, c_{12}, \ldots, c_{n-1,2})$, denotata come $d(\mathbf{c_1}, \mathbf{c_2})$, è il numero di componenti nei quali i due vettori differiscono.

Si verifica che la distanza tra due vettori corrisponde al peso della loro differenza:

$$d(\mathbf{c_i}, \mathbf{c_j}) = w(\mathbf{c_i} - \mathbf{c_j}) \tag{2.11}$$

E' quindi possibile analizzare la distanza di Hamming fra tutte le possibili copie di vettori appartenenti ad un codice. La distanza minima d_{min} corrisponde alla minore di queste distanze:

$$d_{min} = min\{d(\mathbf{c_i}, \mathbf{c_j});\ \mathbf{c_i}, \mathbf{c_j} \in C_b;\ \mathbf{c_i} \neq \mathbf{c_j}\} \tag{2.12}$$

Un importante teorema deriva da tali equazioni:

Teorema 2.4.1. *La distanza minimima di Hamming d_{min} di un codice di blocco lineare $C_b(n, k)$ corrisponde al peso minimo delle parole di codice non nulle di tale codice:*

$$d_{min} = min\{w(\mathbf{c_m});\ \mathbf{c_m} \in C_b;\ \mathbf{c_m} \neq \mathbf{0}\} \tag{2.13}$$

La distanza minima determina la *capacità di correzione*. Per un codice lineare vale infatti

$$2t + 1 \leq d_{min} \leq 2t + 2 \longrightarrow t = \lfloor \frac{d_{min} - 1}{2} \rfloor \tag{2.14}$$

dove t rappresenta il massimo numero di simboli che possono essere corretti.

2.5 Codici ciclici

I codici ciclici rappresentano una classe importante dei codici a blocco lineari, caratterizzata dal fatto di essere facilmente implementabili utilizzando logica sequenziale o shift register.
Dato un vettore di n componenti $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1}) \in \mathbb{F}_q^n$, la rotazione a destra di questo genera un vettore differente. Pertanto i rotazioni portano al vettore $\mathbf{c}^{(i)} = (c_{n-i}, c_{n-i+1}, \ldots, c_{n-1}, c_0, \ldots, c_{n-i-1})$.

Definizione 2.5 (Codice ciclico)**.** Un codice a blocco lineare è detto ciclico se, per ognuna delle sue parole di codice, la sua i-esima rotazione corrisponde ad un'altra parola di codice.

2.5.1 Rappresentazione polinomiale delle parole di codice

E' possibile rappresentare in forma polinomiale nella variabile x una parola di codice $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ associando ad ogni elemento una potenza di x. Si ottiene la scrittura

$$c(x) = c_0 + c_1 x + \ldots + c_{n-1} x^{n-1} \tag{2.15}$$

2.5.2 Polinomio generatore di un codice ciclico

Una parola di codice $\mathbf{c}$ ruotata a destra di i posizioni può essere rappresentata dalla seguente espressione polinomiale:

$$c^{(i)}(x) = c_{n-i} + c_{n-i+1} x + \ldots + c_{n-i-1} x^{n-1} \tag{2.16}$$

La relazione tra il polinomio ruotato di i posizioni $c^{(i)}(x)$ e l'originale codice polinomiale $c(x)$ ha la forma

$$x^i c(x) = q(x)(x^n + 1) + c^{(i)}(x) \tag{2.17}$$

dove $q(x) = c_{n-i} + c_{n-i+1} x + \ldots + c_{n-1} x^{i-1}$. Risulta che il codice polinomiale $c^{(i)}(x)$ è il resto della divisione del polinomio $x^i c(x)$ per $x^n + 1$:

$$c^{(i)}(x) = x^i c(x) \bmod (x^n + 1) \tag{2.18}$$

denotando con mod l'operazione di resto della divisione tra polinomi.

2.6 Codici polinomiali

Definizione 2.6 (Codici polinomiali)**.** Un codice ciclico è detto *polinomiale* quando ad esso è associato un polinomio di grado $n-k$ detto *polinomio generatore* del codice $g(x)$ tale che, indicato con $m(x)$ il polinomio associato al messaggio da codificare, la parola di codice corrispondente è ottenibile mediante il prodotto

$$c(x) = m(x)g(x) = \tag{2.19}$$
$$= (m_0 + m_1 x + \ldots + m_{k-1}x^{k-1})(1 + g_1 x + \ldots + g_{n-k-1}x^{n-k-1} + x^{n-k})$$

Una proprietà fondamentale del polinomio generatore $g(x)$ è di essere un divisore di $x^n + 1$. Ciò si traduce nel fatto che un polinomio di grado $n-k$ che sia anche un divisore di $x^n + 1$ può generare un codice lineare ciclico $C_{cyc}(n, k)$.

Capitolo 3

Codici BCH

I codici BCH formano una sottoclasse dei codici ciclici con capacità di correzione di più errori. Furono creati da A. Hocquenghem nel 1959 ed indipendentemente da R. C. Bose e D. K. Ray-Chaudhuri nel 1960. Essi, comunque, inventarono degli algoritmi di codifica ma non le corrispettive decodifiche.
Il vantaggio principale dei codici BCH è la ridotta complessità degli algoritmi di decodifica, dei quali il più utilizzato è quello a sindrome. Tale sistema è infatti implementabile a livello hardware senza molta difficoltà, permettendone l'utilizzo con dispositivi di dimensioni, potenze e costi ridotti.

3.1 Introduzione

I codici BCH sono un tipo di codici lineari ciclici polinomiali i quali, come si è visto nel precedente capitolo, hanno la proprietà di avere parole di codice che ruotate danno altre parole di codice e che sono il prodotto della moltiplicazione tra il polinomio generatore $g(x)$ di grado $n-k$ e il messaggio $m(x)$.
Per ogni numero intero positivo $m \geq 3$ e $t \leq 2^{m-1}$ esiste un codice binario BCH $C_{BCH}(n,k)$ con le seguenti proprietà [6]:

Lunghezza	$n = 2^m - 1$
Bit di parità	$n - k \leq mt$
Distanza di Hamming minima	$dmin \geq 2t + 1$
Errori correggibili	t errori in una parola di codice

Il polinomio generatore di un codice BCH di tipo (n,k) viene specificato mediante le sue radici nel campo di Galois GF(2^m). Sia α un elemento primitivo di GF(2^m), ovvero una radice n-esima dell'unità in tale campo. Si ha che il polinomio generatore $g(x)$ di un codice BCH con capacità di correzione di t errori corrisponde al polinomio di grado minimo di GF(2^m) tale che abbia α, α^2,$\alpha^3, \cdots, \alpha^{2t}$ come sue radici, ovvero che $g(\alpha^i) = 0, 1 \leq$

$i \leq 2t$.
Detto $\phi_i(x)$ il *polinomio minimo* di α^i, si può definire $g(x)$ come il *minimo comune multiplo* di $\phi_1(x), \phi_2(x), \cdots \phi_{2t}(x)$:

$$g(x) = mcm\{\phi_1(x), \phi_2(x), \cdots \phi_{2t}(x)\} \tag{3.1}$$

per via delle ripetizioni della radici coniugate risulta sufficiente trovare il minimo comune multiplo dei polinomi minimi con indice dispari:

$$g(x) = mcm\{\phi_1(x), \phi_3(x), \cdots \phi_{2t-1}(x)\} \tag{3.2}$$

dal momento che il grado massimo di tali polinomi è m il grado di $g(x)$ è al più pari a mt, da qui la condizione $n - k \leq mt$.

3.2 Codifica BCH dello Standard DVB-T2

Come già visto nel primo capitolo, lo Standard per la televisione digitale DVB-T2 nella parte di codifica FEC (*Forward Error Correction*) prevede l'uso, nell'ordine, di un encoder BCH seguito da un encoder LDPC e da un Bit Interleaver. I pacchetti in entrata all'intero blocco vengono detti `BBFRAME`, mente il flusso di uscita è composto da `FECFRAME`. Come visibile in figura 1.2, i `BBFRAME` hanno lunghezza K_{bch}. Con la codifica BCH $N_{bch} - K_{bch}$ bit di parità detti `BCHFEC` vengono appesi in coda al pacchetto, che assume pertanto una lunghezza di $N_{bch} = K_{ldpc}$ bit. Al termine della codifica LDPC la lunghezza N_{ldpc} dell'intero pacchetto, detto `FECFRAME`, potrà essere di 64800 o di 16200 bit, e si parla rispettivamente di `FECFRAME` normale e breve. Nelle tabelle 3.1 e 3.2 sono riportati i possibili valori dei parametri sopra elencati, in dipendenza dal tipo di `FECFRAME` prodotto. Viene indicata anche la capacità di correzione di errori massima del codice BCH adottato, t, e il LDPC code rate, dato dal rapporto K_{ldpc}/N_{ldpc}.
In [1] si legge che il polinomio generatore dell'encoder BCH con capacità di correzione di t errori si ottiene moltiplicando i primi t polinomi riportati in tabella 3.3 per $N_{ldpc} = 64800$ e in tabella 3.4 per $N_{ldpc} = 16200$, in accordo con l'equazione (3.2).
Secondo [1], inoltre, i passaggi necessari per la codifica di un frame in banda base, rappresentato come bit di un messaggio $M = (m_{K_{bch}-1}, \ldots, m_1, m_0)$ sono i seguenti:

- Si moltiplica il polinomio del messaggio $m(x) = (m_{K_{bch}-1}x^{K_{bch}-1} + m_{K_{bch}-2}x^{K_{bch}-2} + \ldots + m_1 x + m_0)$ per $x^{N_{bch}-K_{bch}}$.
- Si divide $x^{N_{bch}-K_{bch}}m(x)$ per il polinomio generatore. Sia $d(x) = (d_{N_{bch}-K_{bch}-1}x^{N_{bch}-K_{bch}-1} + \ldots + d_1 x + d_0)$ il resto della divisione:
$$d(x) = x^{N_{bch}-K_{bch}}m(x) \bmod g(x)$$

- La parola di codice I si costruisce quindi come segue:

$$\begin{aligned} I &= (i_0, i_1, \cdots, i_{N_{bch}-1}) = \\ &= (m_{K_{bch}-1}, m_{K_{bch}-2}, \ldots, m_1, m_0, d_{N_{bch}-K_{bch}-1}, \ldots, d_1, d_0) \end{aligned}$$

  Il polinomio equivalente associato alla parola di codice è:

$$c(x) = x^{N_{bch}-K_{bch}} m(x) + d(x)$$

Si rende evidente il fatto che i codici BCH fanno parte dei codici sistematici. Infatti, il termine $x^{N_{bch}-K_{bch}} m(x)$ equivale ad una traslazione del messaggio $m(x)$ che non viene alterato. Il termine $d(x)$, di lunghezza $N_{bch} - K_{bch}$, corrisponde invece agli $n-k$ bit di parità che vengono accodati al messaggio per costituire la parola di codice.

3.3 Realizzazione

Si discuterà ora come possa essere implementato in maniera hardware l'algoritmo di codifica BCH appena esaminato.

3.3.1 LFSR

Il *Linear Feedback Shift Register*, il cui acronimo è LFSR, è un particolare tipo di shift register (registro a scorrimento) nel quale ad ogni ciclo di funzionamento i valori memorizzati possono essere determinati dalla operazione logica di or esclusivo (xor) tra uno o più bit del registro stesso. Il suo funzionamento è deterministico (ma può essere utilizzato anche per realizzare generatori di numeri pseudo-casuali) e lo stato futuro dei bit memorizzati dipende interamente dallo stato presente. I bit del registro che vanno ad influenzare lo stato successivo attraverso le funzioni di xor vengono detti *tap*. La posizione dei tap mappa nell'hardware un polinomio caratteristico espresso in modulo 2 associato all'LFSR. Ad esempio, in figura 3.1, è visibile un LFSR di Galois al quale è associato il polinomio $p(x) = 1+x+x^2+x^5$. Si è cercato di rendere graficamente evidente che il polinomio viene rappresentato dal coefficiente di grado zero (sempre unitario) a quello di grado maggiore dalla sinistra alla destra della catena di Flip-Flop. Vi è una connessione in retroazione laddove il relativo coefficiente polinomiale è non nullo.

Un LFSR viene detto a *lunghezza massima* quando il suo polinomio è primitivo, e questo gli consente di passare attraverso tutte le $2^n - 1$ configurazioni possibili, dove n è il numero di registri dell'LFSR. La configurazione fatta di soli zeri è infatti proibita in quanto, per via del comportamento delle porte xor, non rende possibile cambiare il valore dei bit.
Esistono due tipi di realizzazioni degli LFSR, gli LFSR di *Fibonacci* e gli LFSR di *Galois*. Si analizza qui brevemente solo la seconda tipologia, in

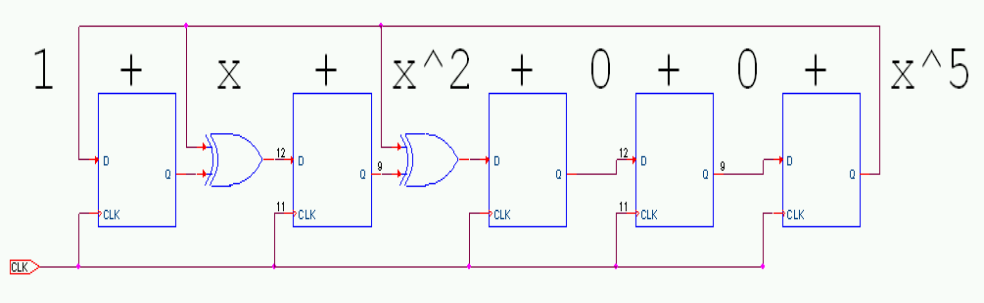

Fig. 3.1: Realizzazione circuitale di un LFSR di Galois che implementa il polinomio $p(x) = 1 + x + x^2 + x^5$.

quanto il cuore dell'implementazione dell'encoder BCH è proprio un LFSR di Galois.

LFSR di Galois
I registri LFSR di Galois si prestano particolarmente ad essere usati in sistemi di codifica che operano con la teoria dei campi finiti (detti anche campi di Galois) binari, come lo è il BCH, e prendono il loro nome dal matematico francese Évariste Galois.
In tali LFSR i bit contenuti nelle locazioni di memoria che non sono tap, all'esecuzione di un nuovo ciclo, vengono trasferiti direttamente nella posizione successiva (ovvero a destra, in riferimento alla figura 3.1). Dove vi è un tap, invece, i bit vengono xorati assieme all'ultimo bit (connesso in retroazione) prima di essere memorizzati nella posizione successiva. Il valore del bit di feedback influenza pertanto l'andamento del registro. Infatti, quando questo è nullo, tutti i bit del registro vengono trasferiti a destra senza alcun cambiamento. Quando invece vale uno i bit di tap, dopo la porta xor, invertono il proprio valore logico e sono quindi memorizzati nella posizione adiacente.

3.3.2 Implementazione su FPGA

Riassumendo, è necessario implementare un algoritmo che, dato un messaggio informativo $m(x)$ in ingresso e un polinomio generatore di un codice BCH $g(x)$ generi in uscita una parola di codice $c(x)$. Come visto, la parola di codice è esprimibile come

$$c(x) = x^{N_{bch}-K_{bch}} m(x) + d(x)$$

dove $d(x)$ è

$$d(x) = x^{N_{bch}-K_{bch}} m(x) \bmod g(x)$$

In maniera equivalente, si può dire che il messaggio in entrata deve essere trasferito pari pari in uscita e che a questo devono seguire gli $N_{bch} - K_{bch}$

bit di $d(x)$. E' pertanto necessario un circuito che calcoli il resto di una divisione polinomiale; i Linear Feedback Shift Register di Galois ne permettono facilmente la realizzazione. La differenza sostanziale con l'impiego classico dell'LFSR sta nell'aggiunta di un input seriale costituito dal messaggio da codificare e nella scelta del bit da inserire ad ogni ciclo di funzionamento nella parola di codice di uscita. Una semplice versione del circuito così modificato è visibile in figura 3.2. Si nota la presenza di due multiplexer; `MUX1` condiziona il bit di feedback mentre `MUX2` seleziona l'uscita. Il segnale di selezione dei due multiplexer è comune, e dipende dalle due fasi di funzionamento:

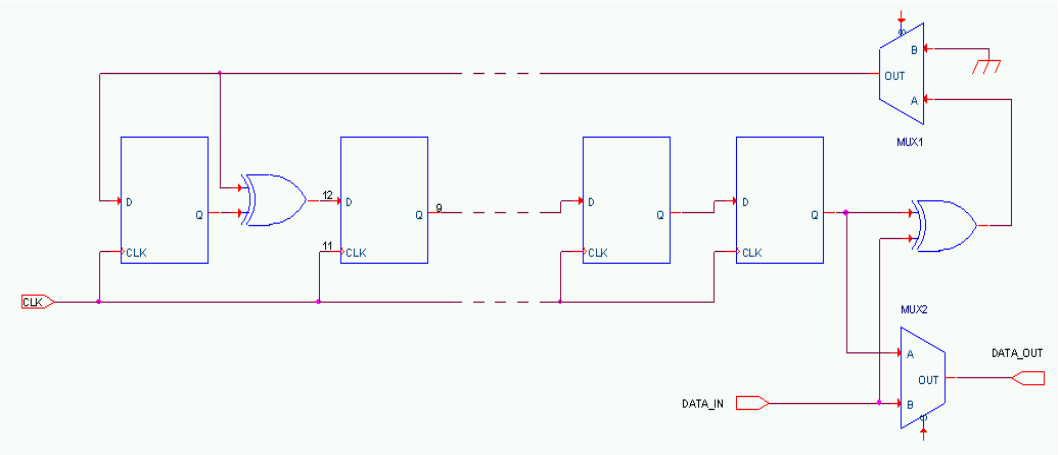

Fig. 3.2: Realizzazione circuitale minima di un encoder BCH.

- ***Fase di acquisizione del messaggio*** All'inizio di una codifica i bit memorizzati nell'LFSR sono tutti zeri. Ad ogni ciclo di funzionamento viene fornito in input un nuovo bit del messaggio, che viene xorato con l'ultimo bit del registro. Il risultato di tale operazione viene poi diffuso nel resto del circuito, e ne condiziona il funzionamento nella maniera già vista precedentemente. Come dato di uscita, attraverso il multiplexer `MUX2`, viene passato il bit d'ingresso. Quando i bit del messaggio da codificare sono esauriti, ovvero dopo k cicli, si passa alla seconda fase di funzionamento.

- ***Fase di output dei bit di parità*** Tale fase ha il compito di accodare alla parola di codice gli $n-k$ bit di parità. Grazie alla proprietà dell'LFSR di Galois, tali bit corrispondono agli $n-k$ bit memorizzati dal registro al termine della prima fase di funzionamento, che vanno perciò estratti ordinatamente, uno per ciclo. Questo shiftaggio in uscita si ottiene fornendo uno zero logico costante in input attraverso `MUX1`, e passando l'ultimo bit dell'LFSR direttamente all'output. In tal modo si è privato l'LFSR della retroazione e questo si trova a funzionare

come un semplice shift register. Al termine di questa seconda fase il registo è completamente azzerato, e pertanto pronto ad accettare un nuovo messaggio in entrata.

LDPC Code Rate	k_{bch}	$n_{bch} = k_{ldpc}$	t	$n_{bch} - k_{ldpc}$	n_{ldpc}
1/2	32208	32400	12	192	64800
3/5	38688	38880	12	192	64800
2/3	43040	43200	10	160	64800
3/4	48408	48600	12	192	64800
4/5	51648	51840	12	192	64800
5/6	53840	54000	10	160	64800

Tabella 3.1: Parametri di codifica per `FECFRAME` normali ($N_{ldpc} = 64800$).

LDPC Code Rate ideale	k_{bch}	$n_{bch} = k_{ldpc}$	t	$n_{bch} - k_{ldpc}$	LDPC Code Rate effettivo	n_{ldpc}
1/4	3072	3240	12	168	1/5	16200
1/2	7032	7200	12	168	4/9	16200
3/5	9552	9720	12	168	3/5	16200
2/3	10632	10800	12	168	2/3	16200
3/4	11712	11880	12	168	11/15	16200
4/5	12432	12600	12	168	7/9	16200
5/6	13152	13320	12	168	37/45	16200

Tabella 3.2: Parametri di codifica per `FECFRAME` brevi ($N_{ldpc} = 16200$).

$g_1(x)$	$1 + x^2 + x^3 + x^5 + x^{16}$
$g_2(x)$	$1 + x + x^4 + x^5 + x^6 + x^8 + x^{16}$
$g_3(x)$	$1 + x^2 + x^3 + x^4 + x^5 + x^7 + x^8 + x^9 + x^{10} + x^{11} + x^{16}$
$g_4(x)$	$1 + x^2 + x^4 + x^6 + x^9 + x^{11} + x^{12} + x^{14} + x^{16}$
$g_5(x)$	$1 + x + x^2 + x^3 + x^5 + x^8 + x^9 + x^{10} + x^{11} + x^{12} + x^{16}$
$g_6(x)$	$1 + x^2 + x^4 + x^5 + x^7 + x^8 + x^9 + x^{10} + x^{12} + x^{13} + x^{14} + x^{15} + x^{16}$
$g_7(x)$	$1 + x^2 + x^5 + x^6 + x^8 + x^9 + x^{10} + x^{11} + x^{13} + x^{15} + x^{16}$
$g_8(x)$	$1 + x + x^2 + x^5 + x^6 + x^8 + x^9 + x^{12} + x^{13} + x^{14} + x^{16}$
$g_9(x)$	$1 + x^5 + x^7 + x^9 + x^{10} + x^{11} + x^{16}$
$g_{10}(x)$	$1 + x + x^2 + x^5 + x^7 + x^8 + x^{10} + x^{12} + x^{13} + x^{14} + x^{16}$
$g_{11}(x)$	$1 + x^2 + x^3 + x^5 + x^9 + x^{11} + x^{12} + x^{13} + x^{16}$
$g_{12}(x)$	$1 + x + x^5 + x^6 + x^7 + x^9 + x^{11} + x^{12} + x^{16}$

Tabella 3.3: Polinomi BCH per `FECFRAME` normali ($N_{ldpc} = 64800$).

$g_1(x)$	$1 + x + x^3 + x^5 + x^{14}$
$g_2(x)$	$1 + x^6 + x^8 + x^{11} + x^{14}$
$g_3(x)$	$1 + x + x^2 + x^6 + x^9 + x^{10} + x^{14}$
$g_4(x)$	$1 + x^4 + x^7 + x^8 + x^{10} + x^{12} + x^{14}$
$g_5(x)$	$1 + x^2 + x^4 + x^6 + x^8 + x^9 + x^{11} + x^{13} + x^{14}$
$g_6(x)$	$1 + x^3 + x^7 + x^8 + x^9 + x^{13} + x^{14}$
$g_7(x)$	$1 + x^2 + x^5 + x^6 + x^7 + x^{10} + x^{11} + x^{13} + x^{14}$
$g_8(x)$	$1 + x^5 + x^8 + x^9 + x^{10} + x^{11} + x^{14}$
$g_9(x)$	$1 + x + x^2 + x^3 + x^9 + x^{10} + x^{14}$
$g_{10}(x)$	$1 + x^3 + x^6 + x^9 + x^{11} + x^{12} + x^{14}$
$g_{11}(x)$	$1 + x^4 + x^{11} + x^{12} + x^{14}$
$g_{12}(x)$	$1 + x + x^2 + x^3 + x^5 + x^6 + x^7 + x^8 + x^{10} + x^{13} + x^{14}$

Tabella 3.4: Polinomi BCH per `FECFRAME` brevi ($N_{ldpc} = 16200$).

Capitolo 4

Implementazione Hardware

4.1 Analisi del componente

In figura 4.1 è visibile una rappresentazione a livello di componente dell'Encoder BCH realizzato. Prima di procedere con l'analisi è indispensabile comprendere la funzione degli ingressi e delle uscite. I segnali di ingresso sono:

- **DATA_IN** è l'ingresso seriale del messaggio da codificare.
- **NEW_DATA_IN** quando è posto a livello alto indica la disponibilità di un nuovo bit di messaggio.
- **MOD_SEL** permette di selezionare la modalità operativa dell'encoder tra quelle previste dallo standard DVB-T2. E' un bus di 4 bit che consente 16 configurazioni di ingresso, ma di queste ne sono usate solamente 13, che sono sufficienti a selezionare tutte le possibili modalità di encoding.
- **CE** è il segnale di *Component Enable* che abilita il funzionamento dell'encoder.
- **RST** quando asserito comanda il reset del componente. In tale evenienza l'LFSR e i contatori vengono azzerati e l'encoder viene predisposto per una nuova acquisizione dati.
- **CLK** rappresenta il segnale di clock del sistema.

I segnali di uscita dell'encoder invece sono:

- **DATA_OUT** è l'uscita seriale della parola codificata.
- **NEW_DATA_OUT** quando è posto a livello alto indica la disponibilità di un nuovo bit in uscita.

- **READY** segnala la disponibilità dell'encoder ad acquisire nuovi bit di ingresso.

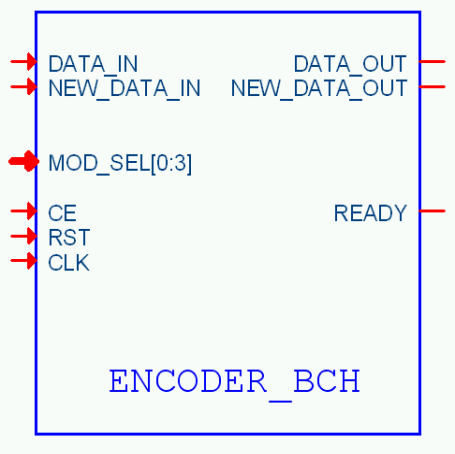

Fig. 4.1: Rappresentazione a livello di componente dell'Encoder BCH.

4.2 Analisi del codice

Il codice VHDL dell'encoder BCH è suddiviso in 10 processi. Si analizzano brevemente qui di seguito tali processi suddividendoli in base al tipo di funzione operata.

4.2.1 Processi di abilitazione del funzionamento

L'encoder BCH è stato realizzato come una macchina a stati molto semplice, essendo infatti provvista di due soli stati, che corrispondono alle due fasi di funzionamento esaminate nel capitolo precedente:

- *get_data* è lo stato iniziale, nel quale l'encoder acquisisce i k bit del messaggio da codificare che vengono passati simmetricamente in uscita.
- *put_data* l'encoder non acquisisce ulteriori dati e fornisce in uscita gli $n - k$ bit di ridondanza della parola di codice.

A svolgere la funzione di controllo dello stato dell'encoder è il processo `P2`. La sua esecuzione viene comandata ad ogni fronte di salita del clock di sistema ma prosegue solo nel caso in cui siano asseriti `CE` ed un importante segnale di controllo interno, *enable_cmd*, del quale si esamina ora l'origine. Si indicherà d'ora in poi con ciclo di funzionamento dell 'encoder un'esecuzione completa dello shift dell'LFSR. Si ha un ciclo di funzionamento solamente se *enable_cmd* è asserito; il suo comandamento avviene ad opera del processo `P3` ed è dipendente dallo stato presente dell'encoder:

- Nello stato *get_data* si hanno k cicli di funzionamento, uno per ogni bit d'ingresso acquisito. Pertanto in tale stato è l'ingresso `NEW_DATA_IN`, che segnala la disponibilità di un nuovo bit del messaggio, a fornire lo stimolo per l'esecuzione di un nuovo ciclo.

- *N*ello stato *put_data* invece il segnale *enable_cmd* è sempre asserito, permettendo così l'esecuzione degli $n - k$ cicli di funzionamento dell'encoder in sincronia con il clock di sistema.

Il codice del processo `P3` è riportato qui di seguito.

```
1  P3: process(CE, RST, stato_pres, NEW_DATA_IN)
2  begin
3    if CE = '1' AND ((stato_pres = get_data AND NEW_DATA_IN = '1')
4            OR (stato_pres = put_data) OR RST = '1') then
5
6      enable_cmd <= '1';
7    else
8      enable_cmd <= '0';
9    end if;
10 end process;
```

Codice 4.1: Processo di abilitazione del funzionamento.

Risulta chiaro che `P3` asserisce *enable_cmd* oltre che nelle situazioni esposte sopra anche nel caso in cui vi sia una richiesta di reset (`RST` alto), e comunque solo se il componente è abilitato (`CE` alto).

4.2.2 Processi di controllo di stato

Come è stato accennato sopra, il processo responsabile del controllo dello stato dell'Encoder è `P2`. Esso è provvisto di due contatori, *count* e *count1*, associati rispettivamente agli stati *get_data* e *put_data*. Questi, nell'ordine, hanno l'importante funzione di tenere il conto dei bit di messaggio ricevuti in ingresso e dei bit di parità forniti in uscita. Ad ogni ciclo di funzionamento dell'encoder viene incrementato il contatore relativo allo stato presente, tranne nel caso che questo abbia già raggiunto specifici valori, in qual caso viene comandato il cambiamento di stato modificando il segnale rappresentante lo stato futuro. E' quindi il processo `P1`, eseguito ad ogni fronte di salita del clock, ad aggiornare effettivamente lo stato presente. I valori specifici che comandano il cambiamento di stato sono pari a $k-1$ per *get_data* e a $n-k-2$ nel caso di *put_data*. Tali valori permettono di eseguire, a seconda del caso, ancora uno o due cicli di funzionamento prima del cambiamento di stato, garantendo così i k e gli $n - k$ cicli necessari alla codifica di una parola di codice di lunghezza n.

```
1  P1: process(CLK)
2  begin
3    if rising_edge(CLK) then
4    stato_pres <= stato_fut;
5    end if;
6  end process;
7
8  P2: process(CLK)
9  begin
10   if rising_edge(CLK) then
11   if enable_cmd = '1' then
12
13     if RST = '1' then
14       stato_fut <= get_data;
15       count <= (OTHERS => '0');
16       count1 <= (OTHERS => '0');
17     else
18       case stato_pres is
19       when get_data =>
20         if count = k_1 then
21           stato_fut <= put_data;
22           count <= (OTHERS => '0');
23         else
24           count <= count+1;
25         end if;
26       when put_data =>
27         if count1 = n_k_2 then
28           stato_fut <= get_data;
29           count1 <= (OTHERS => '0');
30         else
31           count1 <= count1+1;
32         end if;
33       when others =>
34       end case;
35     end if;
36
37   end if;
38   end if;
39 end process;
```

Codice 4.2: Processi di controllo di stato.

I valori k ed n, come già spiegato, non sono fissi ma variano in relazione alla modalità di encoding selezionata. Ad occuparsi di fornire i valori corretti è il processo `P10`.

4.2.3 Processi di esecuzione del ciclo di funzionamento dell'LFSR

La struttura dell'LFSR implementato è stata descritta nel precedente capitolo. Il codice ad esso relativo è contenuto nel processo `P5`, riportato qui di seguito. Si nota che per avere un ciclo di funzionamento è necessario che il segnale *enable_cmd* sia alto. Il processo si occupa, nel caso ci fosse una richiesta di reset, di azzerare il contenuto dei 192 Flip-Flop costituenti l'LFSR, mentre in condizioni di funzionamento normali esegue il semplice shifting da un Flip-Flop al successivo oppure lo shifting con operazione di XOR con il segnale *xoring_data*, a seconda che il relativo coefficiente del polinomio generatore adottato (espresso da *mem_out*) sia zero oppure uno. Il processo `P4` ha invece l'importante ruolo di passare all'LFSR il segnale *xoring_data*, che viene mandato all'ingresso delle porte logiche xor poste in corrispondenza dei *tap* dell'LFSR. Come è stato spiegato, *xoring_data* ha sempre valore nullo nella fase di svuotamento dell'LFSR, ovvero nello stato *put_data*. Durante lo stato *get_data*, invece, *xoring_data* corrisponde allo xor tra il bit del messaggio d'ingresso e il contenuto dell'ultimo Flip-Flop utilizzato dell'LFSR. Si noti che la posizione di tale Flip-Flop è relativa al polinomio generatore usato, o equivalentemente dalla modalità di encoding adottata.

```
1  P5: process(CLK)
2  begin
3    if rising_edge(CLK) then
4    if enable_cmd = '1' then
5      if RST = '1' then
6      lfsr <= (OTHERS => '0');
7      else
8        lfsr(0) <= xoring_data;
9
10       for i in 0 to 190 loop
11       if (mem_out(i) = '0') then
12         lfsr(i+1) <= lfsr(i);
13       else
14         lfsr(i+1) <= lfsr(i) XOR xoring_data;
15       end if;
16     end loop;
17
18     end if;
19   end if;
20   end if;
21 end process;
```

Codice 4.3: Processo che esegue il ciclo di funzionamento dell'LFSR.

4.2.4 Processi di controllo del segnale DATA_OUT

Attraverso DATA_OUT viene fornita in uscita la parola di codice. Il processo P7 che lo controlla è equivalente ad un multiplexer (figura 4.2) comandato dallo stato presente. Quando questo è *get_data* si passa direttamente in uscita il bit del messaggio di ingresso, quando invece lo stato è *put_data* viene selezionato il segnale *lfsr_last*. Quest'ultimo rappresenta semplicemente il contenuto dell'ultimo flip-flop *utile* dell'LFSR, la cui posizione dipende dal polinomio generatore utilizzato. Spetta al processo P6 il compito di selezionare il flip-flop corretto.

```
1  P7: process(CLK)
2  begin
3    if rising_edge(CLK) then
4
5    case stato_pres is
6      when get_data =>
7      DATA_OUT(0)<= DATA_IN;
8      when put_data =>
9      DATA_OUT(0) <= lfsr_last;
10     when others =>
11   end case;
12
13   end if;
14 end process;
```

Codice 4.4: Processo di selezione del segnale di uscita.

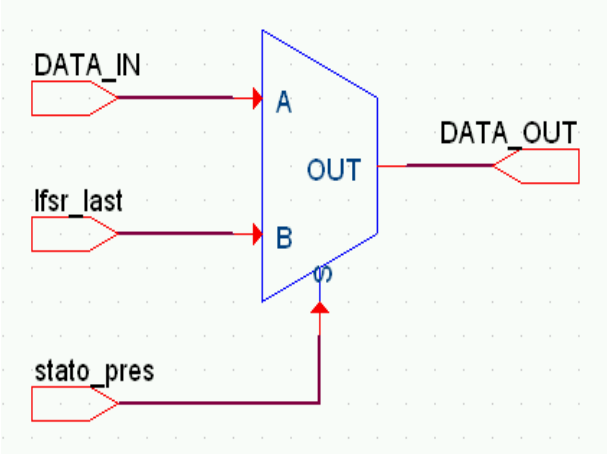

Fig. 4.2: Rappresentazione di P7 come multiplexer.

4.2.5 Processo di controllo del segnale NEW_DATA_OUT

NEW_DATA_OUT ha il compito di segnalare, quando viene posto al valore logico uno, la disponibilità di un nuovo bit di uscita sulla linea DATA_OUT. Il processo P8 (codice 4.5) che lo controlla è equivalente ad un latch che campiona, sul fronte di salita, il segnale proveniente da un multiplexer opportunamente comandato.

```
1  P8: process(CLK)
2  begin
3    if rising_edge(CLK) then
4    if enable_cmd = '1' AND RST = '0' AND
5       ((stato_pres = get_data) OR (stato_pres = put_data)) then
6      NEW_DATA_OUT <= '1';
7    else
8      NEW_DATA_OUT <= '0';
9    end if;
10   end if;
11 end process;
```

Codice 4.5: Processo di controllo del segnale NEW_DATA_OUT.

4.2.6 Processo di controllo del segnale READY

Il segnale READY rimane alto finchè l'encoder è nella fase di acquisizione del messaggio per segnalare al blocco ad esso precedente che è disponibile a ricevere un nuovo bit di ingresso. Col passaggio nella fase di svuotamento dell'LFSR il segnale viene abbassato, onde evitare la perdita di informazione in ingresso. Ad occuparsi del controllo di READY è il processo P9, equivalente ad un multiplexer a due entrate comandato dallo stato presente.

4.2.7 Processo di selezione dei parametri di codifica

Si sono già viste le diverse modalità operative dell'Encoder BCH previste dallo standard DVB-T2 e i parametri ad esse associati. Il processo P10 ha il semplice compito di fornire al processo di controllo P2 i segnali *k_1* e *n_k_2*, rappresentanti rispettivamente i valori $k-1$ ed $n-k-2$ associati alla modalità di encoding utilizzata.

4.2.8 Memoria distribuita

La realizzazione dell'encoder BCH ha reso necessario tenere memorizzati nella FPGA i coefficienti dei tre polinomi generatori utilizzati nelle diverse modalità di codifica. Non essendo la quantità di memoria richiesta grande e visto il semplice tipo di utilizzo si è ricorsi all'uso di una memoria ROM distribuita provvista solamente di un ingresso di selezione della modalità e di una uscita rappresentante i coefficienti.

Capitolo 5

Matlab

5.1 Introduzione

Per poter verificare, attraverso le simulazioni, che il comportamento dell'Encoder BCH sia corretto si è reso necessario confrontare i risultati di tali simulazioni con quelli prodotti da un altro encoder, ovviamente di uguali parametri e messaggio d'ingresso, il cui comportamento sia ideale e quindi privo di errori. Il comportamento ideale è ottenibile mediante qualsiasi linguaggio di programmazione software, che consente l'uso di veloci algoritmi di codifica. Per la sua attitudine all'implementazione di simili algoritmi si è scelto di utilizzare il programma *Matlab* per la realizzazione dell'encoder BCH e di altri tre programmi importanti, descritti brevemente qui di seguito:

- `poli_gen.m` utilizzando i polinomi contenuti in [1] riportati nelle tabelle 3.3 e 3.4 calcola, mediante un opportuno algoritmo, i tre polinomi generatori utilizzati dalla codifica BCH dello standard DVB-T2. Tale programma è stato realizzato per la lunghezza e il rischio di commettere errori dovuto allo svolgimento manuale delle operazioni di moltiplicazioni tra polinomi. I polinomi generatori così ricavati, espressi dalla sequenza dei loro coefficienti, sono stati ricopiati nel file `rom_iniz.coe` che inizializza la memoria ROM distribuita dell'encoder BCH, e sono inoltre utilizzati dall'encoder scritto con *Matlab*.

- `data_generator.m` genera un file di testo contenente un flusso di zeri e uni rappresentante il messaggio da codificare, la cui lunghezza, dipendente dalla modalità di codifica utilizzata, è selezionabile.

- `encoder_BCH.m` è l'equivalente, scritto nel linguaggio di programmazione *Matlab*, dell'Encoder BCH realizzato per FPGA.

- `data_compare.m` confronta i file di testo contenenti le parole di codice prodotte dai due encoder e ne determina l'uguaglianza.

Vengono riportati di seguito i codici sorgente commentati dei programmi realizzati, ad esclusione del primo, per la sua lunghezza e per l'importanza limitata.

5.2 Programma di generazione del messaggio da codificare

Il codice del programma `data_generator.m` è riportato in codice 5.1. Dopo avere creato il file di testo `data_in`, predisponendolo per la scrittura, l'applicazione esegue il ciclo *for* principale k volte. Il parametro k va quindi impostato manualmente per ottenere messaggi della lunghezza voluta. Ad ogni esecuzione del ciclo viene generato un bit casuale che viene quindi convertito in carattere e copiato sul file. Al ciclo segue poi la chiusura del file d'uscita.

```
1  % data_generator.m
2
3  clear all;
4  close all;
5
6  % apertura del file
7  fout = fopen('data_in', 'w');
8
9  % lunghezza del messaggio da codificare
10 k = 32208;
11
12
13 for i = 1 : k
14     % generazione casuale di un bit
15     bit = randint(1, 1, [0 1]);
16     % conversione da numero a carattere
17     % e scrittura su file
18     if bit == 1
19         fwrite(fout, '1', 'char');
20     else
21         fwrite(fout, '0', 'char');
22     end
23 end
24
25
26 % chiusura del file
27 fclose(fout);
```

Codice 5.1: File `data_generator.m`.

5.3 Programma che esegue la codifica BCH

Il linguaggio *Matlab* è provvisto di una funzione che esegue la codifica BCH di un vettore dati i parametri n e k e un vettore contenente i coefficienti del polinomio generatore utilizzato. Tuttavia, non la si è potuta usare in quanto opera solamente con codici BCH regolari. Si è così resa necessaria la scrittura di un Encoder BCH software, realizzato nel file `encoder_BCH.m` (codice 5.2). Tale encoder ricalca il modello hardware simulando con un vettore l'utilizzo di un LFSR mentre le variabili *xoring_data* e *data_out* sono le corrispettive degli omonimi segnali VHDL.

```
1  % encoder_BCH.m
2
3  clear all;
4  close all;
5
6  % apertura dei file
7  fin = fopen('data_in.txt', 'r');
8  fout = fopen('data_out_m.txt', 'w');
9
10 % parametri di codifica
11 n = 32400;
12 k = 32208;
13
14  gen_poly_192 = [1 1 1 0 0 1 0 0 1 1 0 0 0 1 1 0 1 0 0 0 1 1 0
15        1 0 0 1 0 0 1 1 1 1 1 0 0 1 0 0 1 1 0 1 0 0 0 1 1 1 1 1
16        1 0 1 0 0 1 0 0 0 0 1 1 1 1 1 0 0 0 1 1 1 0 0 0 1 1 0 0
17        1 1 0 0 1 0 1 0 1 1 1 0 1 1 1 1 0 1 0 1 1 0 0 1 1 0 0 0
18        1 1 0 0 1 0 0 0 0 1 1 0 1 1 1 1 1 1 0 1 1 0 0 0 1 0 0 0 0
19        0 1 1 0 0 1 0 0 1 0 0 0 0 1 1 1 0 1 1 1 1 1 1 0 0 0 1 0
20        0 1 0 1 0 0 1 0 0 1 1 1 0 0 1 1 1 1 0 1 0 0 1 1 1 0 0 1
21        0 1];
22
23  gen_poly_160 = [1 0 0 0 1 0 1 1 0 0 0 1 0 1 0 0 0 1 0 0 1 1 1
24        0 1 1 0 0 0 1 0 1 1 0 0 1 1 0 1 1 0 0 1 1 0 1 0 1 0 1 0
25        0 1 1 1 1 1 0 0 0 1 0 0 0 1 1 0 0 0 1 1 0 1 1 0 0 0 1 0 1
26        1 1 1 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 1
27        1 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0 0 1 0 0 1 0 1 1 0 0 0 0
28        1 0 0 0 0 1 1 1 0 1 1 0 0 0 1 1 0 0 0 0 0 0 1 1 0 1];
29
30  gen_poly_168 = [1 0 1 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0 0 1 1 0 0
31        0 1 0 0 0 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 1
32        1 0 0 0 1 0 1 0 0 1 0 1 0 1 0 0 1 0 1 1 0 0 0 0 0 1 0 0
33        1 1 1 0 0 0 1 0 1 1 1 0 0 0 1 0 0 1 0 1 1 0 0 1 1 0 1 0
34        0 0 1 1 0 0 1 0 0 1 1 0 1 1 0 0 1 0 1 1 0 0 0 0 1 1 0 0
35        1 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0 0 0
36        0 0 0 1 0 1];
37
38 % selezione del polinomio generatore da utilizzare
39 gen_poly = gen_poly_192;
40
41 lfsr = zeros(2, n-k);
```

```
42 for i = 1 : n
43     % assegnazione dei segnali di controllo
44   if 1<=i && i<=k
45         % lettura del bit del messaggio
46         num = fscanf(fin, '%c', 1);
47         % conversione da carattere in intero
48         if num == '1'
49             data_in = 1
50         else
51             data_in = 0
52         end
53
54         data_out = data_in;
55     xoring_data = xor(lfsr(1, n-k), data_in);
56
57     elseif k<i && i<=n
58         data_out = lfsr(1, n-k);
59         xoring_data = 0;
60     end
61
62   % scrittura su file del bit d'uscita
63     if data_out == 1
64         fwrite(fout, '1', 'char');
65     else
66         fwrite(fout, '0', 'char');
67     end;
68   % ritorno a capo ogni 100 bit
69   if mod(i, 100) == 0
70     fprintf(fout, '\n');
71   end
72
73
74     % ciclo dell'LFSR
75   lfsr(2,1) = xor_data;
76
77   for j = 2 : (n-k)
78     if gen_poly(j) == 0
79       lfsr(2,j) = lfsr(1,j-1);
80     else
81       lfsr(2,j) = xor(lfsr(1,j-1), xoring_data);
82         end
83     end
84
85     % ripulitura dell'array di supporto
86     lfsr(1, 1:n-k) = lfsr(2, 1:n-k);
87
88 end
89
90 % chiusura dei file
91 fclose(fin);
92 fclose(fout);
```

Codice 5.2: File `encoder_BCH.m`.

5.4 Programma di confronto

Per verificare che la parola di codice ottenuta dall'Encoder BCH scritto in VHDL sia corretta si utilizza il file `data_compare.m` (codice 5.3). Nel suo ciclo principale il programma legge un bit dal file di uscita dell'Encoder VHDL e uno dal file d'uscita dell'Encoder *Matlab*, scartando i caratteri di rientro a capo (codisci ASCII 10 e 13) e li confronta. Se essi sono diversi l'esecuzione viene subito interrotta. Se, invece, al termine degli n confronti le due parole di codice risultano uguali viene stampata la riga `uguali = 1` sulla console.

```
% data_compare.m

clear all;
close all;

% apertura dei file
fvhdl = fopen('data_out.txt', 'r');
fmatl = fopen('data_out_m.txt', 'r');

% parametro di codifica
n = 32400;

for i = 1 : n
    % lettura di un bit
    num1 = fscanf(fvhdl, '%c', 1);
    num2 = fscanf(fmatl, '%c', 1);
    % scarta i caratteri di rientro a capo
    while (num1 == char(10) || num1 == char(13))
        num1 = fscanf(fvhdl, '%c', 1);
    end
    while (num2 == char(10) || num2 == char(13))
        num2 = fscanf(fmatl, '%c', 1);
    end
    % confronto dei due bit
    if num1 ~= num2
        % se diversi termina l'esecuzione
        uguali = 0
        break
    end
    % se tutti i bit corrispondono
    % segnala l'uguaglianza
    if i == n
        uguali = 1
    end
end

% chiusura dei file
fclose(fvhdl);
fclose(fmatl);
```

Codice 5.3: File `data_compare.m`.

Capitolo 6

Testbench

6.1 Test del funzionamento

Nel capitolo precedente sono state analizzate tre applicazioni *Matlab* il cui sviluppo si è reso necessario per poter verificare il funzionamento dell'Encoder BCH. Questi programmi vengono utilizzati, assieme al programma di testbench, per simulare il comportamento dell'encoder BCH VHDL e controllarne gli esiti. Per effetture una simulazione di test è necessario svolgere le seguenti azioni:

- Scegliere la modalità di codifica da utilizzare e indicarla nelle applicazioni *Matlab* e nel file di testbench.
- Creare, utilizzando il programma `data_generator.m`, un file contenente i k bit del messaggio da codificare.
- Eseguire la codifica con l'encoder BCH scritto in *Matlab*, che produce il file `data_out_m.txt`.
- Effettuare una simulazione dell'Encoder scritto in VHDL. Il programma utilizzato (*Xilinx ISE 12.1*) da l'opportunità di scegliere fra *Behavioral, Post-Translate, Post-Map* e *Post-Route Simulation*, il cui grado di affidabilità è crescente. La simulazione produce il file `data_out.txt`.
- Confrontare i file di uscita dei due encoder con il programma `data_compare.m`, per verificare se la parola di codice prodotta è la stessa.

6.2 Programma di testbench

I software di simulazione VHDL richiedono un file scritto anch'esso in VHDL, detto file di *testbench*, che ha il compito di fornire le informazioni necessarie alla simulazione che non sono direttamente ricavabili dal codice sotto test. Tali informazioni riguardano infatti fattori come l'andamento nel tempo dei

segnali di ingresso, compreso il clock di sistema. Il testbench realizzato è chiamato `encoder_BCH_tb.vhd` ed è composto di due importanti processi, oltre a quello predisposto alla generazione del clock di sistema. L'encoder BCH opera con grandi flussi di dati e ciò ha reso necessario far sì che il testbench possa leggere e scrivere dei file. Il processo `stim_proc` (codice 6.1) si occupa della lettura dal file generato con *Matlab* del messaggio da codificare e ne pone i bit, uno per ogni ciclo di clock, sulla linea `DATA_IN`, regolando `NEW_DATA_IN` di conseguenza.

```
1  stim_proc: process
2
3    -- dichiarazione delle variabili utilizzate
4    variable l : LINE;
5    variable fBit : BIT_VECTOR (0 downto 0);
6
7  begin
8    -- attesa iniziale
9    wait for 20 * PERIOD;
10
11   -- ciclo di lettura e immissione dati
12   while NOT endfile(inData) loop
13     -- lettura di una linea da file
14     readline(inData, l);
15     -- si alza la linea di segnalazione
16     NEW_DATA_IN <= '1';
17
18     for i in 1 to l'LENGTH loop
19       -- lettura di un bit
20       read(l,fBit);
21       -- copiatura del bit sulla linea DATA_IN
22       DATA_IN <= to_stdlogicvector(fBit) (0);
23       -- attesa di un periodo di clock
24       wait for PERIOD;
25     end loop;
26
27     -- si abbassa la linea di segnalazione
28     NEW_DATA_IN <= '0';
29   end loop;
30 end process;
```

Codice 6.1: Processo di input del file di testbench `encoder_BCH_tb.vhd`.

Il compito del processo `out_proc` (codice 6.2) è invece quello di ricopiare i bit della parola di codice posti dall'encoder sulla linea `DATA_OUT` in un file chiamato `data_out.txt`, inserendo un ritorno a capo ogni 100 bit. Siccome la scrittura avviene per righe, appunto ogni 100 bit, per evitare di escludere possibili dati finali di lunghezza inferiore, si è prevista la scrittura di un'ultima riga la cui lunghezza dipende dalla modalità di encoding utilizzata, e quindi dal valore del segnale `MOD_SEL`.

```
1  out_proc: process(CLK)
2
3    -- dichiarazione delle variabili utilizzate
4    variable lout : LINE;
5    variable fBitOut : BIT_VECTOR (0 downto 0);
6
7  begin
8    if rising_edge(CLK) AND NEW_DATA_OUT = '1' then
9      -- lettura del bit
10   fBitOut := to_bitvector(DATA_OUT)(0 downto 0);
11   -- scrittura su una riga
12   write(lout, fBitOut);
13   i := i+1;
14   -- scrittura su file della riga di 100 bit
15   if i = 100 then
16     writeline(outData, lout);
17     i := 0;
18     j := j+1;
19   -- scrittura di eventuali altri bit rimanenti
20   -- a seconda della modalita' di codifica
21   elsif MOD_SEL = "0001" AND i = 80 AND j = 388 then
22     writeline(outData, lout);
23     i := 0;
24     j := 0;
25   elsif MOD_SEL = "0100" AND i = 40 AND j = 518 then
26     writeline(outData, lout);
27     i := 0;
28     j := 0;
29   elsif MOD_SEL = "0110" AND i = 40 AND j = 32 then
30     writeline(outData, lout);
31     i := 0;
32     j := 0;
33   elsif MOD_SEL = "1000" AND i = 20 AND j = 97 then
34     writeline(outData, lout);
35     i := 0;
36     j := 0;
37   elsif MOD_SEL = "1010" AND i = 80 AND j = 118 then
38     writeline(outData, lout);
39     i := 0;
40     j := 0;
41   elsif MOD_SEL = "1100" AND i = 20 AND j = 133 then
42     writeline(outData, lout);
43     i := 0;
44     j := 0;
45   end if;
46   end if;
47 end process;
```

Codice 6.2: Processo di scrittura del file di uscita.

Capitolo 7

Risultati

7.1 Simulazione dell'Encoder BCH

In questa sezione sono riportate delle schermate tratte dalla simulazione Post-Route dell'Encoder BCH eseguita con il tool *ISim 12.1* della Xilinx, al fine di analizzarne il comportamento nei punti di maggior interesse. Le simulazioni, tranne dove diversamente indicato, sono state eseguite ad una frequenza di clock di circa 110MHz, pari alla frequenza nominale di lavoro dell'encoder inserito nel modulatore DVB-T2.
La figura 7.1 ritrae l'inizio dell'acquisizione di un nuovo messaggio da codificare. Il primo bit di ingresso (un uno logico) è acquisito sul primo fronte di salita del clock quando `NEW_DATA_IN` è asserito. Si nota quindi che l'acquisizione continua alla velocità di un bit per ciclo di clock, ovverosia l'encoder lavora a *full rate*. In figura 7.2 si sono evidenziati i tempi riguardanti l'esecuzione di un ciclo di funzionamento. Il tempo T_{ciclo} è l'intervallo tra l'acquisizione di un bit di ingresso e l'output dello stesso in uscita, ovvero il tempo necessario per il completemento di un ciclo di funzionamento dell'LFSR, mentre T_{seg} è il tempo necessario perché venga alzato `NEW_DATA_OUT`, in modo da segnalare la disponibilità del dato in uscita.

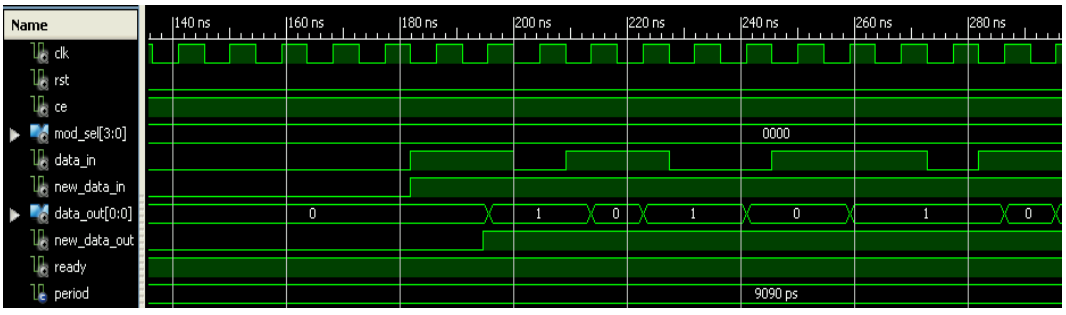

Fig. 7.1: Inizio dell'acquisizione del messaggio.

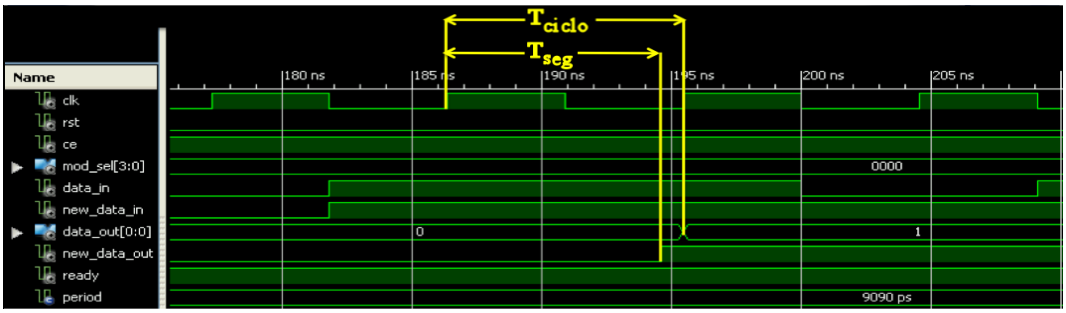

Fig. 7.2: Evidenziazione delle tempistiche dell'encoder.

Il momento in cui termina l'acquisizione del messaggio di ingresso ed inizia la fase di svuotamento dell'LFSR è evidenziato in figura 7.3. Si nota che il segnale `NEW_DATA_OUT` rimane deasserito per un ciclo di clock. Tale ciclo è infatti necessario all'encoder per eseguire il cambiamento di stato e durante tale periodo l'LFSR rimane inattivo. Al fronte di salita successivo del clock l'encoder inizia a lavore nello stato *put_data*. Il segnale `READY` viene abbassato per segnalare l'incapacità dell'encoder di ricevere ulteriori dati in ingresso.

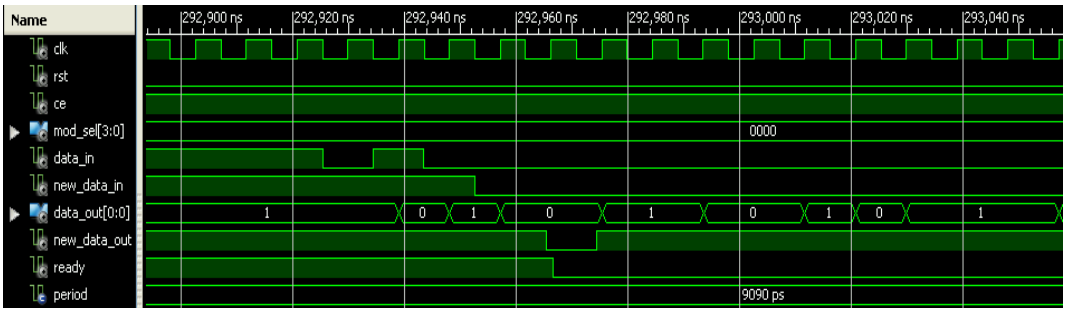

Fig. 7.3: Fine dell'acquisizione del messaggio ed inizio della fase di svuotamento dell'LFSR.

La terza fase più significativa del funzionamento è il termine della fase di svuotamento, visibile in figura 7.4.
Si può notare che l'uscita `READY` viene rialzata in corrispondenza dell'output dell'ultimo bit della parola di codice. Nel caso in esame, in cui la frequenza di clock è di 110 MHz, il primo fronte di salita del clock dopo l'asserzione di `READY` occorre subito dopo la deasserzione di `NEW_DATA_OUT`. Tuttavia, per frequenze di clock maggiori, si ha il primo fronte di salita utile per l'acquisizione mentre l'ultimo bit di output è ancora mantenuto stabile. Ciò potrebbe far pensare che se il blocco che precede l'encoder, visto il segnale `READY` nuovamente alto, cominciasse a trasmettere immediatamente il nuovo messaggio, si avrebbe una perdita di dati da parte dell'encoder. In realtà si

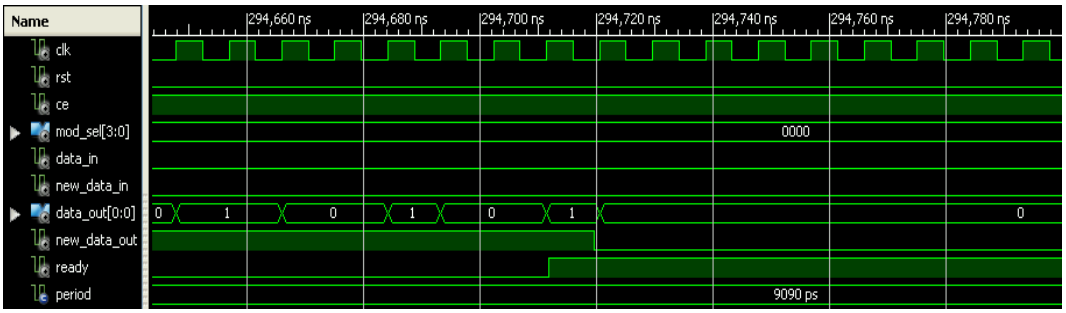

Fig. 7.4: Termine della fase di svuotamento.

è verificato che non si ha nessun malfunzionamento. In figura 7.5, infatti, è visibile il risultato di una simulazione effettuata alla velocità di clock di 133 MHz nella quale si è provveduto ad inviare un nuovo messaggio di input al primo fronte di clock utile. Si nota che per un solo ciclo di clock l'encoder non fornisce dati in uscita (`NEW_DATA_OUT` basso) ma al ciclo successivo comincia a riprodurre il messaggio d'entrata e il funzionamento prosegue correttamente di qui in poi.

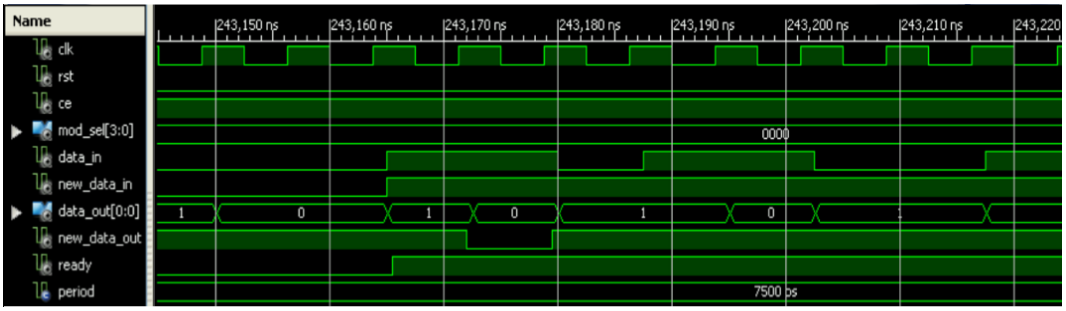

Fig. 7.5: Elaborazione di un nuovo messaggio all'immediato termine della fase di svuotamento ($f_{clock} = 133$MHz).

7.2 Prestazioni e consumo

Le risorse della FPGA Virtex 4 modello XC4VSX25 occupate dall'encoder sono elencate nella tabella 7.1. Le tabelle 7.2 e 7.3 riportano invece le statistiche relative al consumo di potenza alla frequenza di clock di 110 MHz. Come si nota l'utilizzo della FPGA è davvero minimo, attorno all'1%, e molto limitata è anche la potenza richiesta per il funzionamento. Non bisogna scordare che l'Encoder BCH costituisce comunque solo una piccola parte del ben più ampio modulatore DVB-T2, e il progetto nel quale il presente elaborato è inserito consiste nell'implementazione su una singola FPGA dell'intero sistema.

Sommario delle risorse utilizzate			
Logica Utilizzata	Usata	Disponibile	Utilizzo
Numero di Slice Flip Flops	220	20,480	1%
Numero di LUT a 4 input	285	20,480	1%
Numero di Slice occupate	158	10,240	1%
Numero di Slice contenenti solo logica correlata	158	158	100%
Numero di Slice contenenti logica incorrelata	0	158	0%
Numero Totale di LUT a 4 input	307	20,480	1%
Usate come logica	285		
Usate come route-thru	22		
Numero di IOB vincolati	12	320	3%
Numero di BUFG/BUFGCTRL	1	32	3%
Usati come BUFG	1		
Fanout medio delle reti non di Clock	3.88		

Tabella 7.1: Sommario delle risorse utilizzate

7.3 Conclusioni

A lavoro ultimato, l'Encoder BCH realizzato risulta funzionare correttamente in tutte le modalità di codifica previste dallo standard DVB-T2. Il tool *Timing Analyzer* del programma *Xilinx ISE 12.1* riporta un possibile periodo di clock minimo pari a 6.644 ns, approssimativamente corrispondente alla frequenza di 150 MHz. Tale valore è ben superiore alla frequenza di funzionamento nominale dell'encoder, ovvero a quella sufficiente per il corretto funzionamento dell'intero modulatore DVB-T2.
Per quanto i risultati siano positivi, l'encoder così realizzato è sicuramente migliorabile. Ad esempio, con una migliore impostazione della gestione degli stati sarebbe possibile eliminare il periodo di clock durante il quale l'LFSR rimane inattivo che si verifica in entrambe le transizioni tra una fase di funzionamento e l'altra.
Altri possibili migliorie risiedono nelle differenti possibili implementazioni dell'LFSR. Esistono infatti algoritmi di codifica che sfruttano appieno il polinomio generatore del codice BCH adottato, consentendo di ridurre il numero di flip-flop utilizzati, adottando anche tecniche di parallelismo. Tali algoritmi, però, richiedono di esser ben compresi per poter essere impiegati con una determinata codifica, e la teoria matematica che li sostiene è molto avanzata. Inoltre, nel caso dello standard DVB-T2, si usano tre diversi polinomi generatori, che porterebbero quindi all'implementazione di tre diversi LFSR ottimizzati. Tuttavia questi fattori, assieme alla mancanza della necessità di una velocità di funzionamento maggiore, inducono a non soffermarsi troppo su tali tipi di migliorie.

On-Chip	Potenza [W]	Usata	Disponibile	Utilizzo [%]
Clock	0.029	1	–	–
Logica	0.001	307	20480	1.5
Segnali	0.004	328	–	–
IO	0.001	12	320	3.8
DCM	0.000	0	4	0.0
Perdite	0.332	–	–	–

Tabella 7.2: Consumo di potenza alla frequenza di 110 MHz

	Totale	Dinamica	Quiscente
Consumo di potenza [W]	0.367	0.035	0.332

Tabella 7.3: Potenza dissipata alla frequenza di 110 MHz

Bibliografia

[1] Digital video broadcasting (dvb); frame structure channel coding and modulation for a second generation digital terrestrial television broadcasting system (dvb-t2). Technical report, ETSI, 2009.

[2] Digital video broadcasting (dvb); implementation guidelines for a second generation digital terrestrial television broadcasting system (dvb-t2). Technical report, ETSI, 2009.

[3] Marcelo S. Alencar. *Digital Television Systems*. Cambridge University Press, 2009.

[4] Alain Glavieux. *Channel Coding in Communication Networks*. ISTE Ltd, 2007.

[5] Michael Frater John Arnold and Mark Pickering. *Digital Television. Technology and Standards*. John Wiley and Sons, 2007.

[6] Jorge Castiñeira Moreira Patrick Guy Farrell. *Essentials of Error-Control Coding*. John Wiley and Sons, 2006.

[7] Jr. Shu Lin, Daniel J. Costello. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, 1983.